AI 逻辑 tutorial

甄景贤 (King-Yin Yan)

General.Intelligence@Gmail.com

Abstract.

0 Logic-based AI (LBAI)

Main points:

- We would not directly implment logic-based AI, but it serves as a *backdrop* for understanding what are the problems of general AI.
- In this paper we would jump back and forth between the logic-based view and the dynamical state-space view. Knowledge of LBAI is essential to understanding ideas in this paper.

It is feasible to use mathematical logic to emulate human thinking, an approach pioneered by John McCarthy (1927-2011). We have 3 basic operations: deduction, abduction, induction; For details one can refer to 《Computational logic and human thinking》 by Robert Kowalski, 2011. We would not waste time to debate whether LBAI is an adequate model of human thinking; This paper assumes it as the point of departure. It is worth mentioning though, that Kowalski is one of the researchers who laid the theoretical foundations of logic programming, especially Prolog.

In classical logic-based AI, "thinking" is achieved by steps like this:

$$\text{premise} \vdash \text{conclusion} \tag{1}$$

$$\boxed{\text{it was raining this morning}} \vdash \boxed{\text{grass is wet}} \tag{2}$$

That is to say: from some **propositions** we deduce other propositions.

Deduction requires some special propositions known as **rules**, these are propositions containing **variables** such as "x":

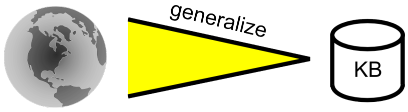
$$\boxed{\text{it is raining at location } x} \wedge \boxed{x \text{ is uncovered}} \vdash \boxed{\text{location } x \text{ is wet}} \tag{3}$$

Rules are like the "fuel" for an inference engine; The engine cannot run without fuel.

Note: The x inside a proposition is like a "hole" in it. We could use **substitution** to place some concrete **objects** into such holes, to make the proposition *complete*. This is a form of

sub-propositional structure, and one way to express it is via **predicate logic**. We don't need to concern with details right now.

LBAI can be viewed as the compression of a world model into a knowledge-base (KB) of logic formulas (that consists of **facts** as well as **rules**):

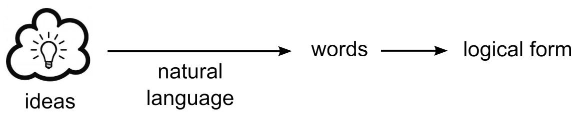


(4)

The world model is *generated* combinatorially from the set of logic formulas, vaguely reminiscent of a "basis" in vector space. The generative process in logic is much more complicated, contributing to its high *compressive* ability on the one hand, and the *complexity* of learning such formulas on the other hand.

1 Bottom-up vs top-down representations

In LBAI the knowledge representation structure is built (*fixed*) from the bottom up (For example, predicate symbols and constant symbols build up propositions, and sets of propositions form theories):



(5)

but is it valid (or profitable) to assume that our mental representations are *isomorphic* to such logical structures? Or drastically different?

The most serious disadvantage of bottom-up representations lies in the difference between **syntactic distance** and **semantic distance**. Suppose propositions are built up from an "alphabet" of atomic concepts, through the use of a multiplication operation such as tensor product. We embed atomic concepts into a vector space, in the manner of the Word2Vec algorithm. Then, using the tensor product, propositions (ie sentences) will be mapped to positions in the tensor-product vector space. Thus we can measure the **distance** between any two propositions. However, this is a **syntactic** distance. For example, *"Don't judge a book by its cover"* and *"Clothes do not make the man"* are superficially very different (syntactically distant) but are semantically close. In a good learning system we need to **generalize** according to semantic distance. The embedding of bottom-up representations usually gives us a discrete space with fractal structure, and the metric defined on such a space is always syntactic.

Humans are good at designing symbolic structures, but we don't know how to design *neural* representations which are more or less opaque to us. Perhaps we could use a neural network acting recurrently on the state vector to **induce** an internal representation of mental space. "*Induced by what*," you ask? By the very structure of the neural network itself. In other words, forcing a neural network to *approximate* the ideal operator R^*.

From an abstract point of view, we require:

- R be an endomorphism: $X \to X$
- R has a learning algorithm: $R \xmapsto{A} R^*$

R would contain all the knowledge of the KB, so we expect it to be "large" (eg. having a huge number of parameters). We also desire R to possess a **hierarchical** structure because hierarchies are computationally very efficient. A multi-layer perceptron (MLP) seems to be a good candidate, as it is just a bunch of numbers (weight matrices W) interleaved by non-linear activation functions:

$$R(\boldsymbol{x}) = \sigma(W_1 \sigma(W_2 ... \sigma(W_L \boldsymbol{x}))) \tag{6}$$

where L is the number of layers. MLPs would be our starting point to explore more design options.

In 1991 Siegelmann and Sontag [2] proved that recurrent neural networks (RNNs) can emulate any Turing machine. In 1993 James Lo [1] proved that RNNs can universally approximate any non-linear dynamical system.

The idea of R as an operator acting on the state is inspired by the "consequence operator" in logic, usually denoted as Cn:

$$\mathrm{Cn}(\Gamma) = \{ \text{ set of propositions that entails from } \Gamma \ \} \tag{7}$$

but the function of R can be broader than logical entailment. We could use R to perform the following functions which are central to LBAI:

- **deduction** – forward- and backward-chaining
- **abduction** – finding explanations
- **inductive learning**

Below, we try to formalize the structure of logic from 2 perspectives:

- Static structure (formulas built from atomic concepts, logic operators, etc)
- Dynamic structure (mechanisms of proof, inference, etc)

2 Static structure of logic

- **truth values** (eg. P(rain tomorrow) = 0.7)
- **propositional structure** (eg. conjunction: $A \wedge B$)
- **sub-propositional structure** (eg. predication: loves(john, mary))
- **subsumption structure** (eg. dog $\subseteq$ animal)

These structures can be "transplanted" to the vector space X via:

- **truth values:** an extra dimension conveying the "strength" of states
- **propositional structure:** eg. conjunction as vector addition,

$$A \wedge B \quad \Leftrightarrow \quad \boldsymbol{x}_A + \boldsymbol{x}_B + \ldots \tag{8}$$

  but we may have to avoid linear dependencies ("clashing") such as:

$$\boldsymbol{x}_3 = a_1 \boldsymbol{x}_1 + a_2 \boldsymbol{x}_2 \tag{9}$$

  This would force the vector space dimension to become very high.

- **sub-propositional structure:** eg. tensor products as composition of concept atoms:

$$\text{loves(john, pete)} \quad \Leftrightarrow \quad \overrightarrow{john} \otimes \overrightarrow{love} \otimes \overrightarrow{pete} \tag{10}$$

- **subsumption structure:** eg. define the **positive cone** C such that

$$\text{animal} \supseteq \text{dog} \quad \Leftrightarrow \quad \overrightarrow{animal} - \overrightarrow{dog} \in C \tag{11}$$

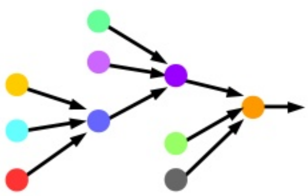
But the more logical structure we add to X, the more it will resemble logic, and this whole exercise becomes pointless. Remember our original goal is to try something different from logic, by *relaxing* what defines a logical structure. So we would selectively add features to X.

3 Dynamic structure of logic

@Andrew: This is where I want to formalize a "logical system". Particularly, the state X has internal structure that I have ignored so far: X should be a **set** of propositions. During deduction, we need to **select** a few propositions from X and try to **match** them with existing logic rules (this is the job of the famous **unifcation** algorithm in logical AI systems). The selection is part of the control variable u (see below). We need to decompose the vector X into some analogue of "propositions", but I don't know how to do it yet. Perhaps elucidating the algebraic form of the logic system will help us design the "vectorization" scheme.

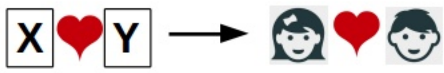
The 2 "pillar" algorithms for deduction in LBAI are:

- **Resolution**: deducing new propositions (conclusions) from existing ones (premises)



(12)

- **Unificaiton**: matching a proposition with variables ("holes") with grounded ("without holes") propositions



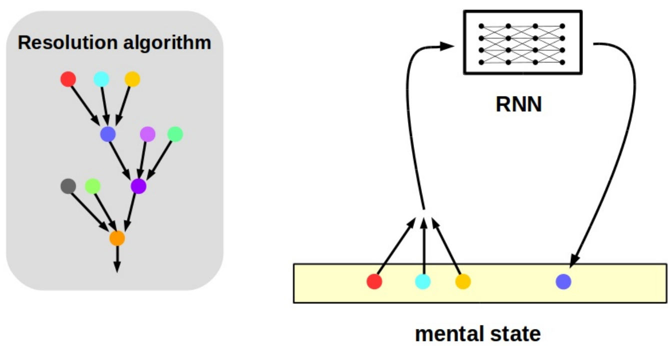
(13)

The algebraization of first-order predicate logic (a logic whose propositions can have internal variables) is a difficult subject, potentially involving Tarski's cylindrical algebra which the author is unfamiliar with.

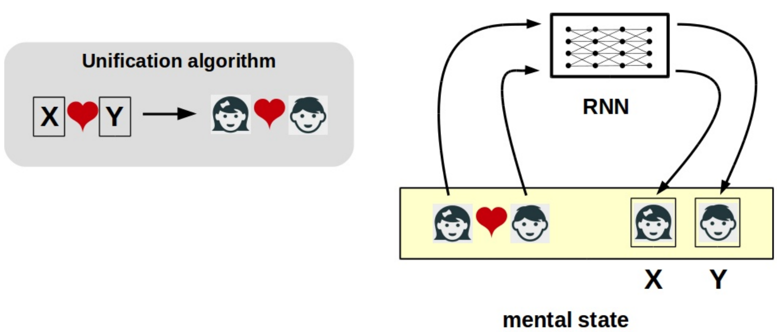
Here we introduce a crucial idea: using an **external memory** to manage the problem of **variable binding**. Recall that a **Turing machine** is just a finite state machine equipped with a **memory tape**; It could be said that the memory tape is what enables the machine to have Turing-complete computing power. Similarly, allowing a **propositional logic** to use an external memory storage for intermediate results, enables it to have the same expressive power as **predicate logic**.

Below are 2 cartoons illustrating how **resolution** and **unification** are performed with the aid of **external memory**:

@Andrew: I want to formalize these operations. It may be more important than formalizing the **static** properties of logic.



(14)



(15)

Example 1: primary-school arithmetic

A recurrent neural network is a *much more powerful* learning machine than a feed-forward network, even if they look the same superficially.

$$\begin{array}{r} 7\;\;3 \\ -_{\triangle}3_{\bullet}7 \\ \hline 3\;\;6 \end{array}$$

As an example, consider the way we perform 2-digit subtraction in primary school. This is done in two steps, and we put a dot on paper to mark "carry-over".

The use of the paper is analogous to the "tape" in a Turing machine – the ability to use short-term memory allows us to perform much more complex mental tasks.

We did a simple experiment to train a neural network to perform primary-school subtraction. The operator is learned easily if we train the two steps *separately*. The challenge is to find an algorithm that can learn **multi-step** operations by itself.

Example 2: variable binding in predicate logic

The following formula in predicate logic defines the "grandfather" relation:

$$\text{grandfather}(X,Z) \longleftarrow \text{father}(X,Y) \wedge \text{father}(Y,Z) \tag{16}$$

We did a simple experiment to train a neural network to perform primary-school subtraction. The operator is learned easily if we train the two steps *separately*. The challenge is to find an algorithm that can learn **multi-step** operations by itself.

References

1. Lo. Dynamical system identification by recurrent multilayer perceptrons. *Proceedings of the 1993 World Congress on Neural Networks*, 1993.
2. Siegelmann and Sontag. Turing computability with neural nets. *Applied Mathematics Letters, vol 4, p77-80*, 1991.